ТИНЬКОФФ

Байткод в JVM

Или как работает JVM

Иванов Артем

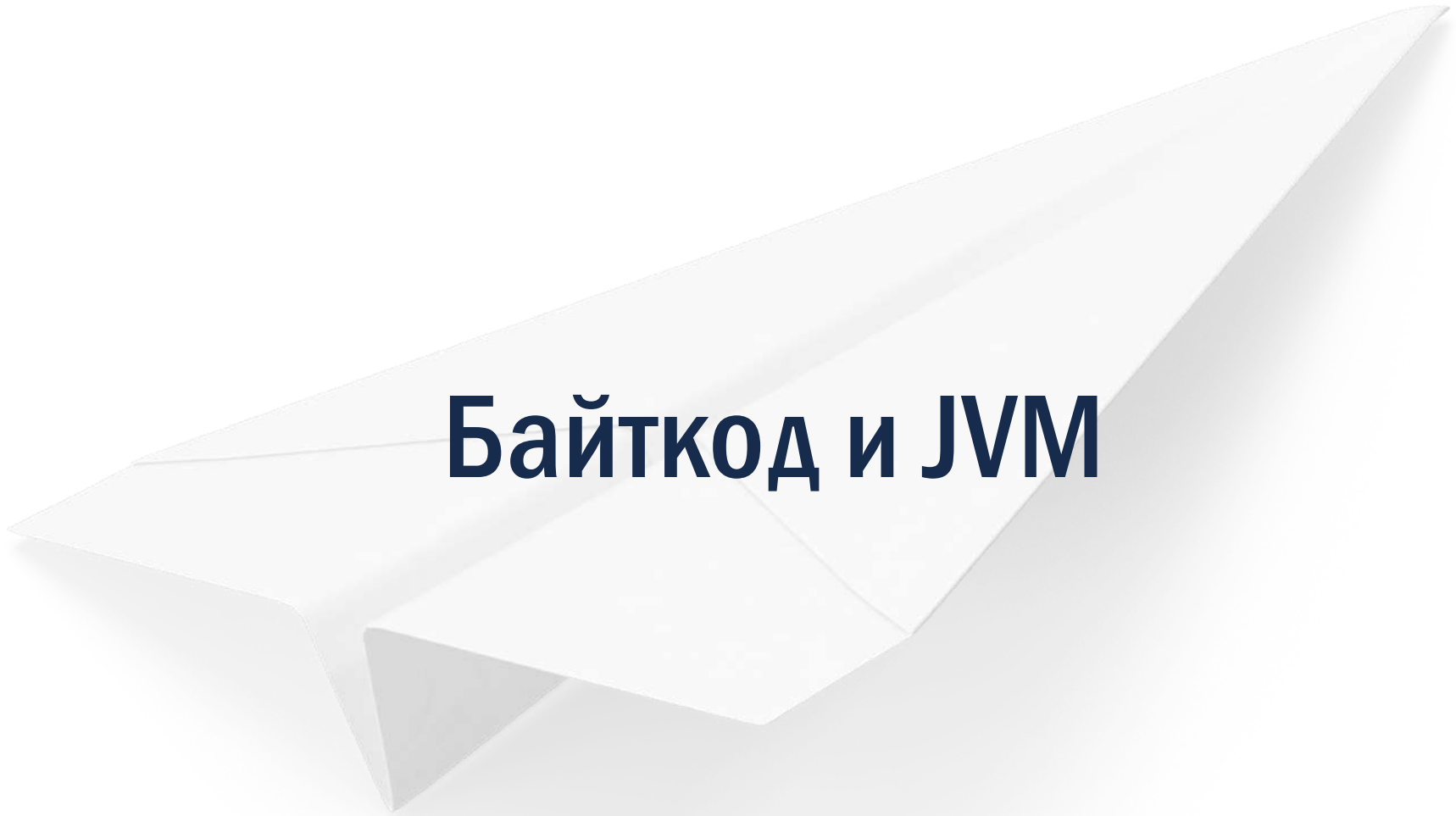
12.12.2023

О себе

- 7 лет занимаюсь разработкой софта
- запускал профессию по Java в Tinkoff и был держателем профессии 1.5 года
- СТО отдела риск технологий
    - в отделе 50 Backend разработчиков на Java/Kotlin
    - Десятки сервисов и сотни тысяч строчек кода на Java/Kotlin
- Пишем и поддерживаем нагруженные, критичные для бизнес процессов компании, Kotlin приложения
    - Миллионы входящих запросов в сутки
    - Десятки миллионов исходящих запросов в сутки
    - Отливаем сотни ТВ данных аналитику

О чем будет лекция

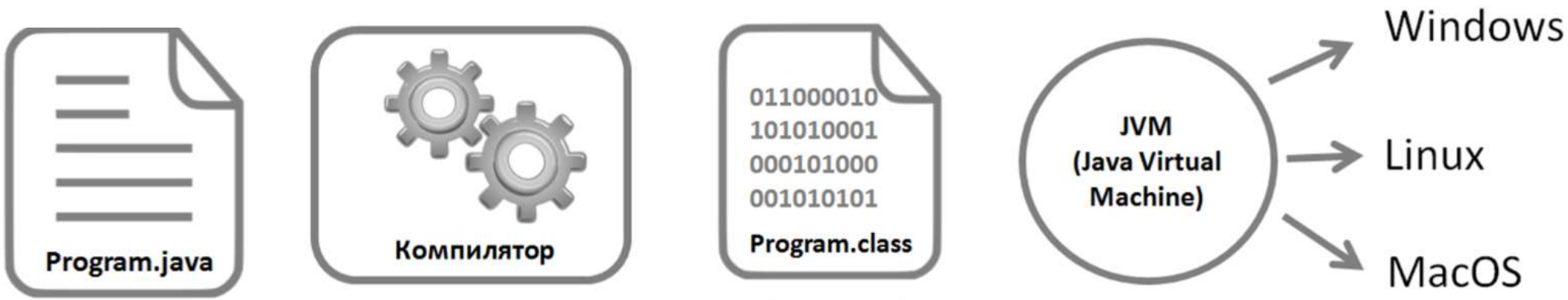
1. Байткод и JVM
2. Спецификация JVM
    1. Этапы загрузки class-файла в JVM
    2. Устройство JVM
    3. Путь от .java файла до машинного кода
3. Устройство class-файла
4. Стековая архитектура
5. Инструкции байткода
6. (бонус!) зачем нужна инструкция nop
7. Обработка ошибок



Байткод и JVM

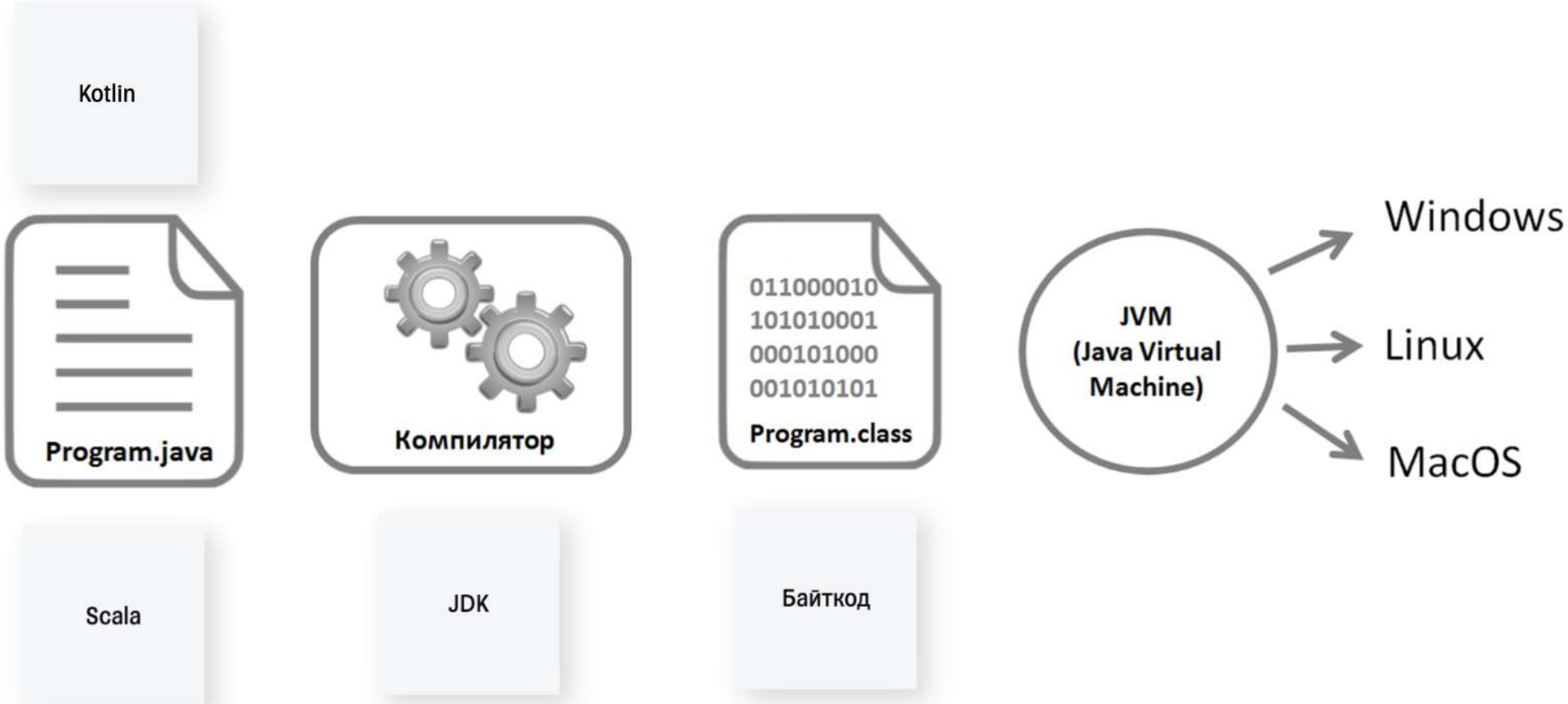
```
public class course.lecture2.example.ExampleMaking {
  public course.lecture2.example.ExampleMaking();
    Code:
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."<init>":()V
       4: return

  public static void main(java.lang.String[]);
    Code:
       0: getstatic     #7                  // Field java/lang/System.out:Ljava/io/PrintStream;
       3: ldc           #13                 // String Hello World!
       5: invokevirtual #15                 // Method java/io/PrintStream.println:(Ljava/lang/String;)V
       8: return

  public java.lang.Integer getValue(java.lang.String);
    Code:
       0: iconst_5
       1: invokestatic  #21                 // Method java/lang/Integer.valueOf:(I)Ljava/lang/Integer;
       4: areturn
}
```

```
cafe babe 0000 003d 002c 0a00 0200 0307
0004 0c00 0500 0601 0010 6a61 7661 2f6c
616e 672f 4f62 6a65 6374 0100 063c 696e
6974 3e01 0003 2829 5609 0008 0009 0700
0a0c 000b 000c 0100 106a 6176 612f 6c61
6e67 2f53 7973 7465 6d01 0003 6f75 7401
0015 4c6a 6176 612f 696f 2f50 7269 6e74
5374 7265 616d 3b08 000e 0100 0c48 656c
6c6f 2057 6f72 6c64 210a 0010 0011 0700
120c 0013 0014 0100 136a 6176 612f 696f
2f50 7269 6e74 5374 7265 616d 0100 0770
7269 6e74 6c6e 0100 1528 4c6a 6176 612f
6c61 6e67 2f53 7472 696e 673b 2956 0a00
1600 1707 0018 0c00 1900 1a01 0011 6a61
7661 2f6c 616e 672f 496e 7465 6765 7201
```


Program.java
Компилятор
011000010
101010001
000101000
001010101
Program.class
JVM
(Java Virtual
Machine)
Windows
Linux
MacOS

JDK

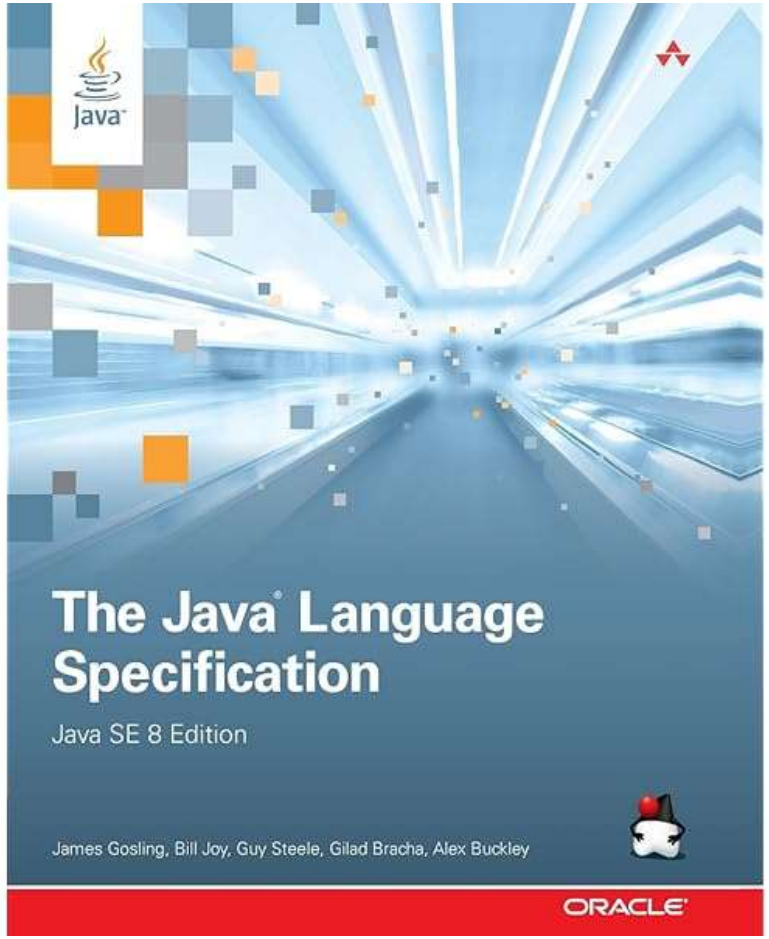
Байткод


Kotlin
Program.java
Компилятор
011000010
101010001
000101000
001010101
Program.class
JVM
(Java Virtual
Machine)
Windows
Linux
MacOS
Scala
JDK
Байткод

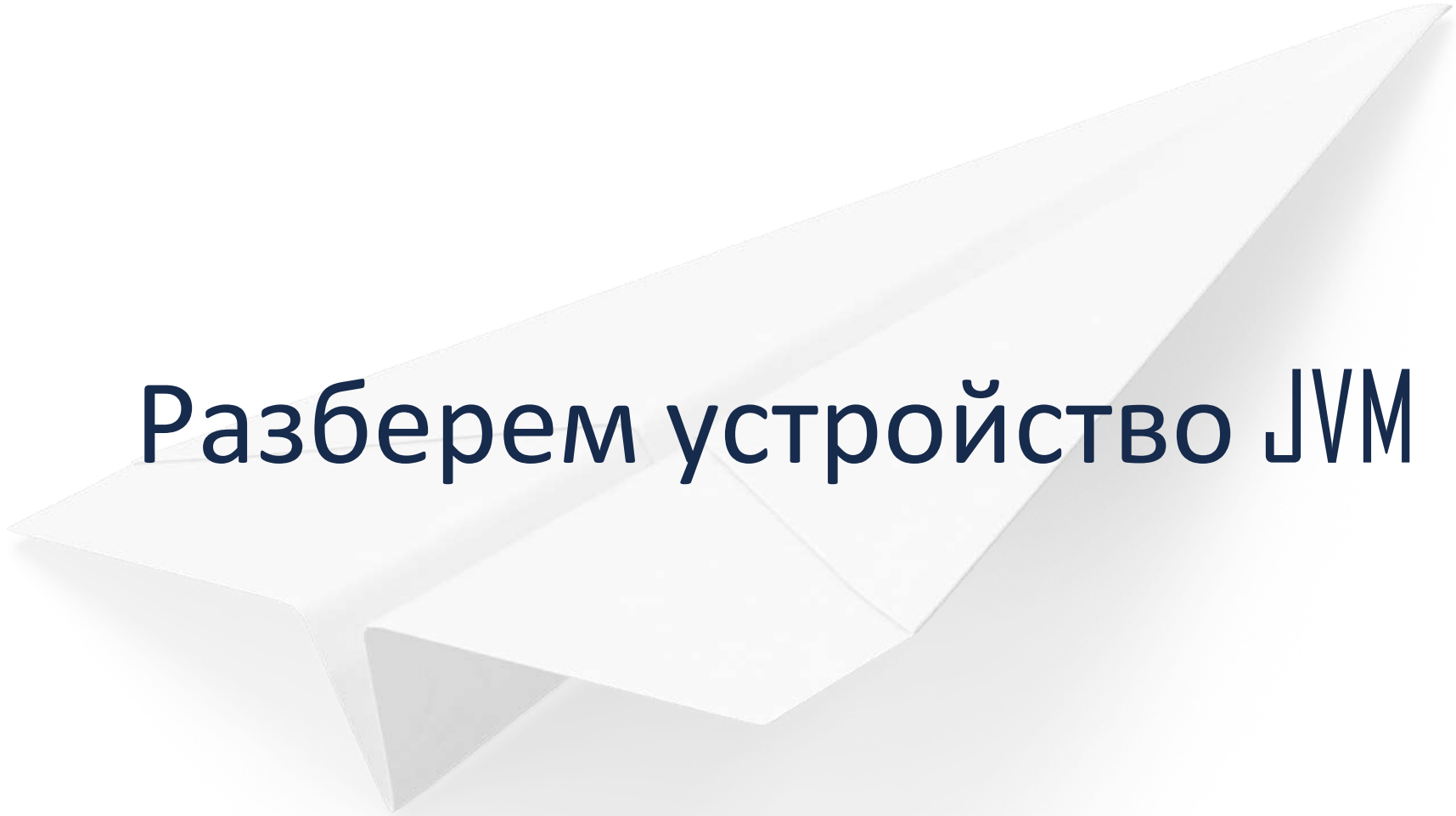
Спецификация JVM

Спецификация JVM

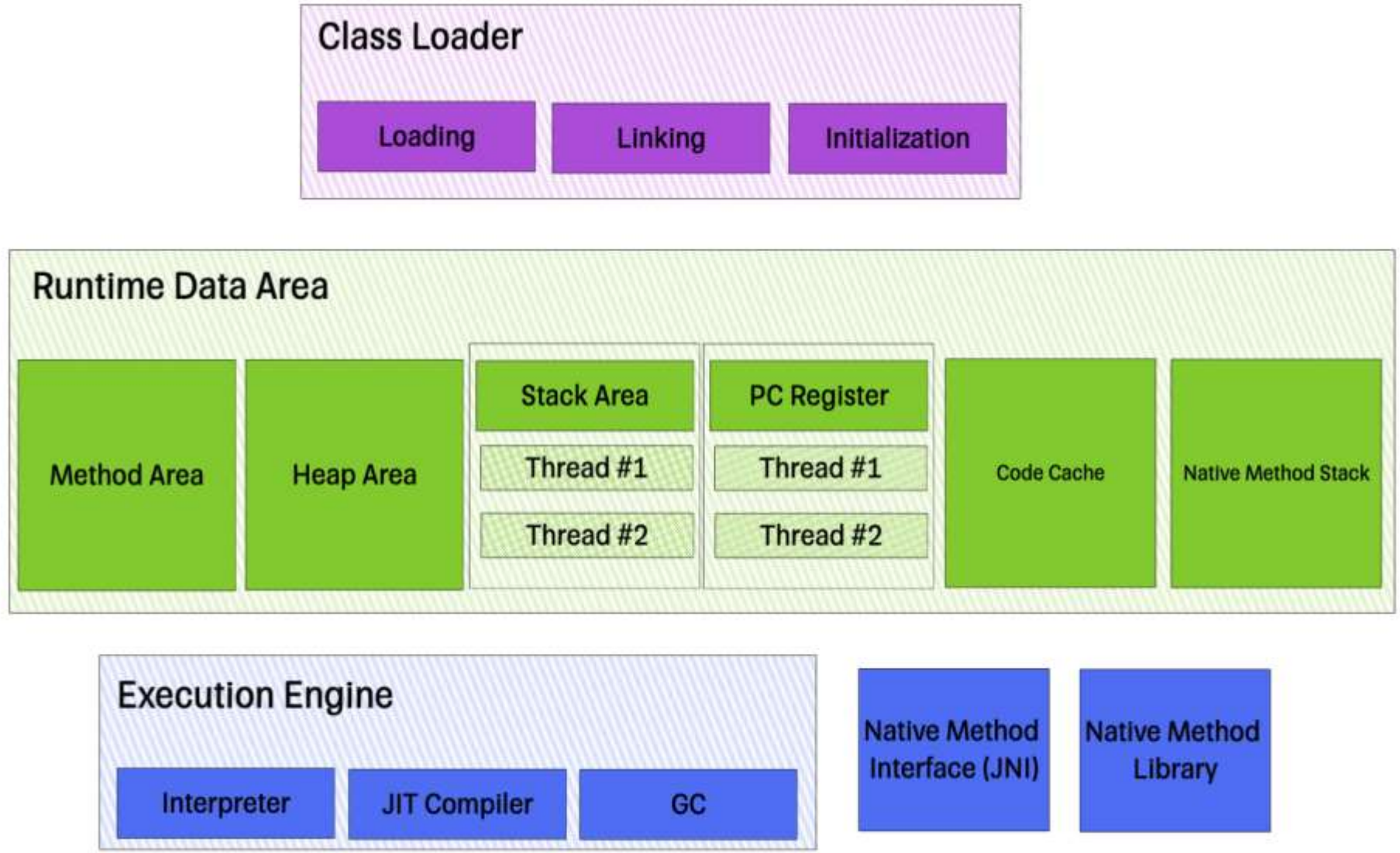
1. Классы и объекты
2. Константный пул
3. Загрузка и проверка классов
4. Инструкции и байткод
5. Стековая архитектура
6. Обработка исключений
7. Управление памятью
8. Многопоточность
9. Загрузка и выполнение байткода
10. Оптимизация и производительность

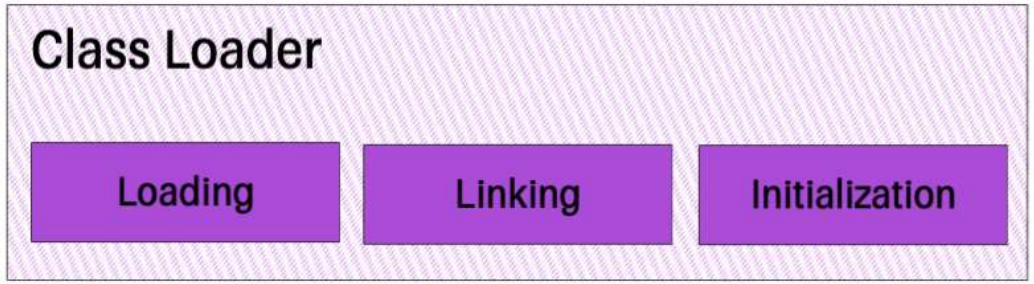


https://docs.oracle.com/javase/specs/jvms/se21/html/

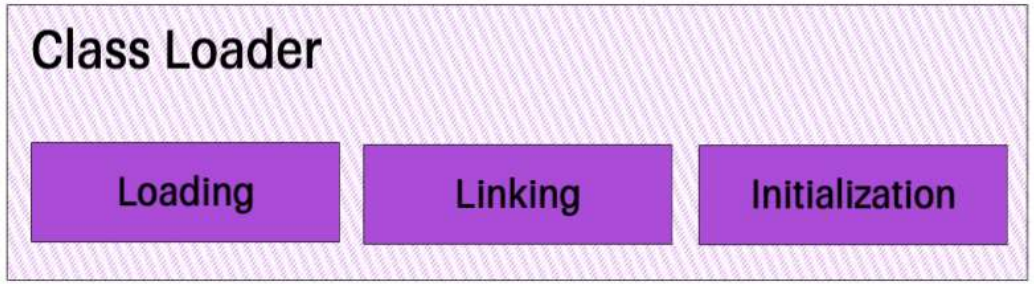


Разберем устройство JVM

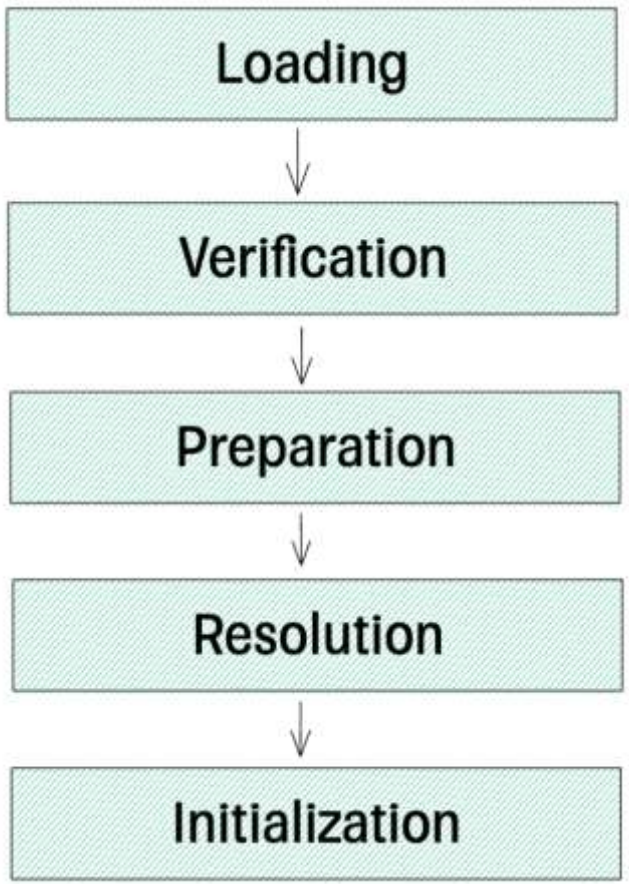

Class Loader
Loading
Linking
Initialization
Runtime Data Area
Method Area
Heap Area
Stack Area
Thread #1
Thread #2
PC Register
Thread #1
Thread #2
Code Cache
Native Method Stack
Execution Engine
Interpreter
JIT Compiler
GC
Native Method Interface (JNI)
Native Method Library

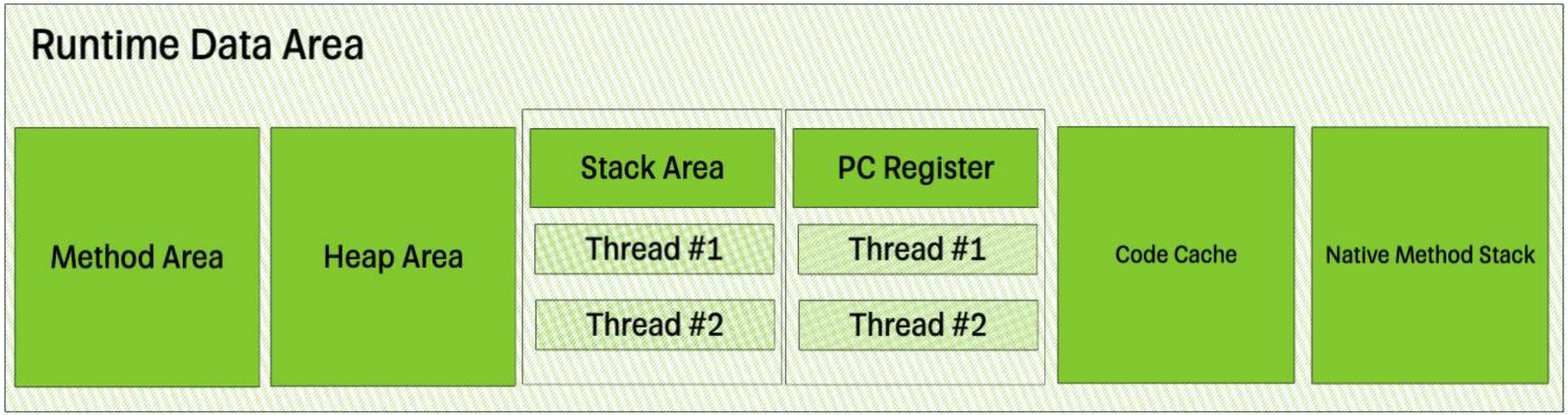



```
cafe babe 0000 003d 002c 0a00 0200 0307
0004 0c00 0500 0601 0010 6a61 7661 2f6c
616e 672f 4f62 6a65 6374 0100 063c 696e
6974 3e01 0003 2829 5609 0008 0009 0700
0a0c 000b 000c 0100 106a 6176 612f 6c61
6e67 2f53 7973 7465 6d01 0003 6f75 7401
0015 4c6a 6176 612f 696f 2f50 7269 6e74
5374 7265 616d 3b08 000e 0100 0c48 656c
6c6f 2057 6f72 6c64 210a 0010 0011 0700
120c 0013 0014 0100 136a 6176 612f 696f
2f50 7269 6e74 5374 7265 616d 0100 0770
7269 6e74 6c6e 0100 1528 4c6a 6176 612f
6c61 6e67 2f53 7472 696e 673b 2956 0a00
1600 1707 0018 0c00 1900 1a01 0011 6a61
7661 2f6c 616e 672f 496e 7465 6765 7201
```

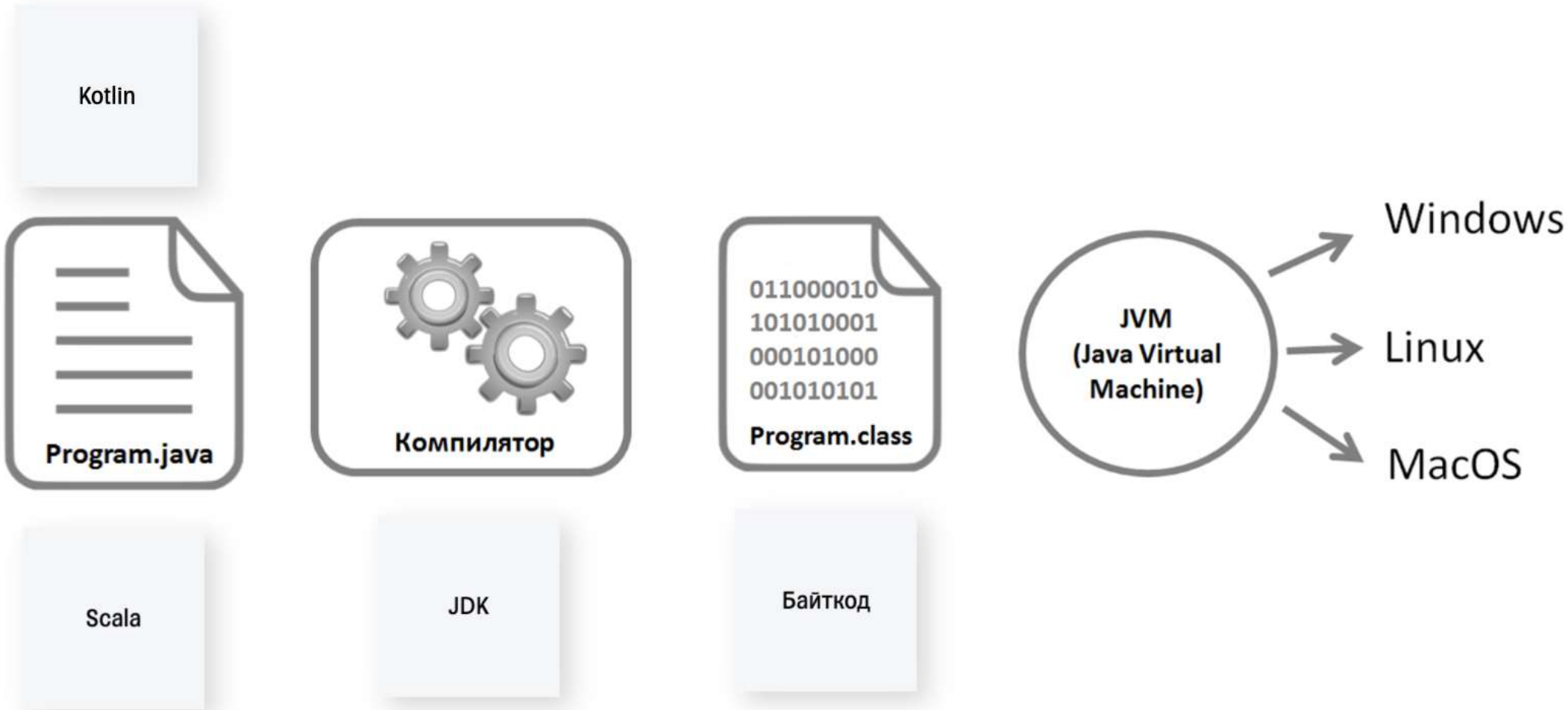


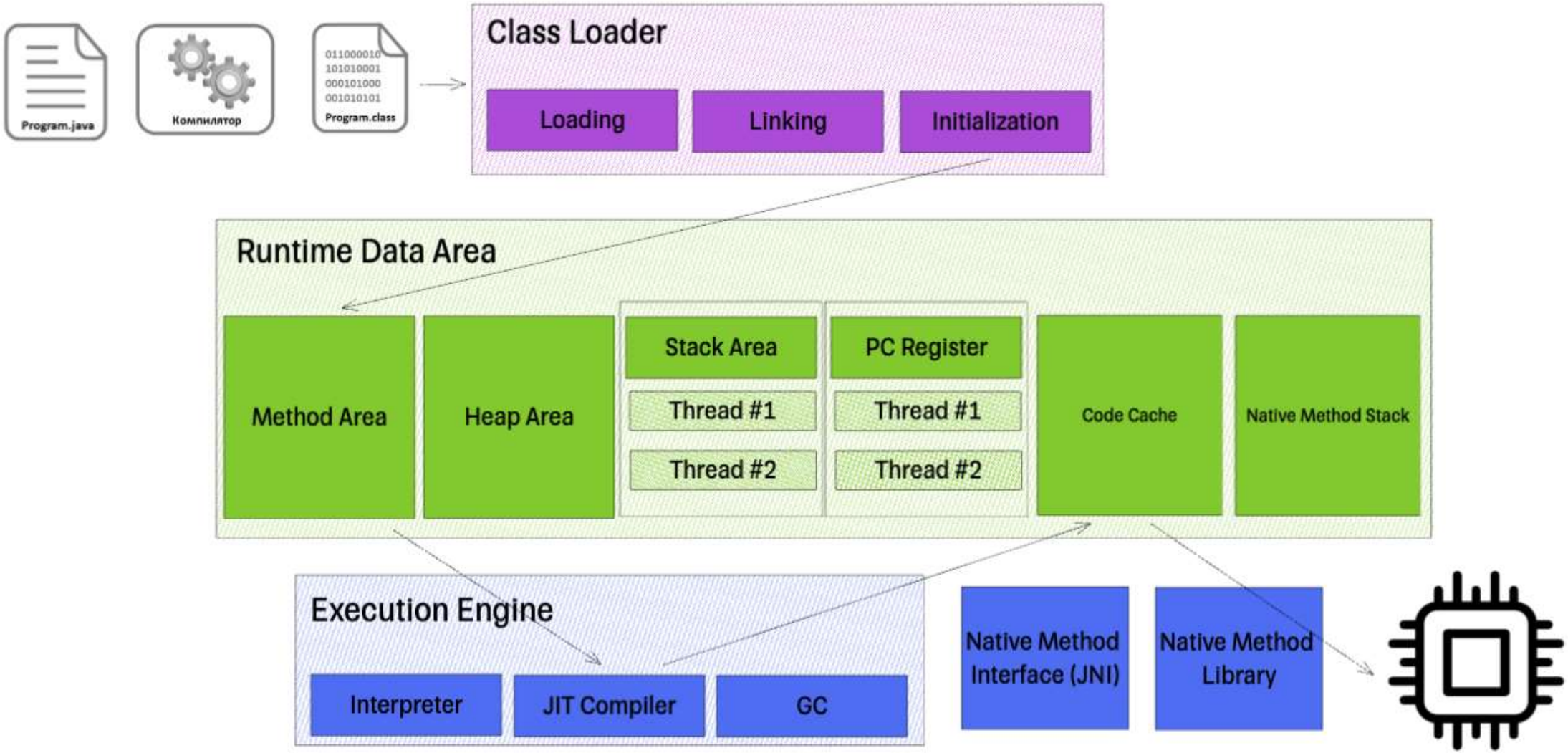

```
cafe babe 0000 003d 002c 0a00 0200 0307
0004 0c00 0500 0601 0010 6a61 7661 2f6c
616e 672f 4f62 6a65 6374 0100 063c 696e
6974 3e01 0003 2829 5609 0008 0009 0700
0a0c 000b 000c 0100 106a 6176 612f 6c61
6e67 2f53 7973 7465 6d01 0003 6f75 7401
0015 4c6a 6176 612f 696f 2f50 7269 6e74
5374 7265 616d 3b08 000e 0100 0c48 656c
6c6f 2057 6f72 6c64 210a 0010 0011 0700
120c 0013 0014 0100 136a 6176 612f 696f
2f50 7269 6e74 5374 7265 616d 0100 0770
7269 6e74 6c6e 0100 1528 4c6a 6176 612f
6c61 6e67 2f53 7472 696e 673b 2956 0a00
1600 1707 0018 0c00 1900 1a01 0011 6a61
7661 2f6c 616e 672f 496e 7465 6765 7201
```

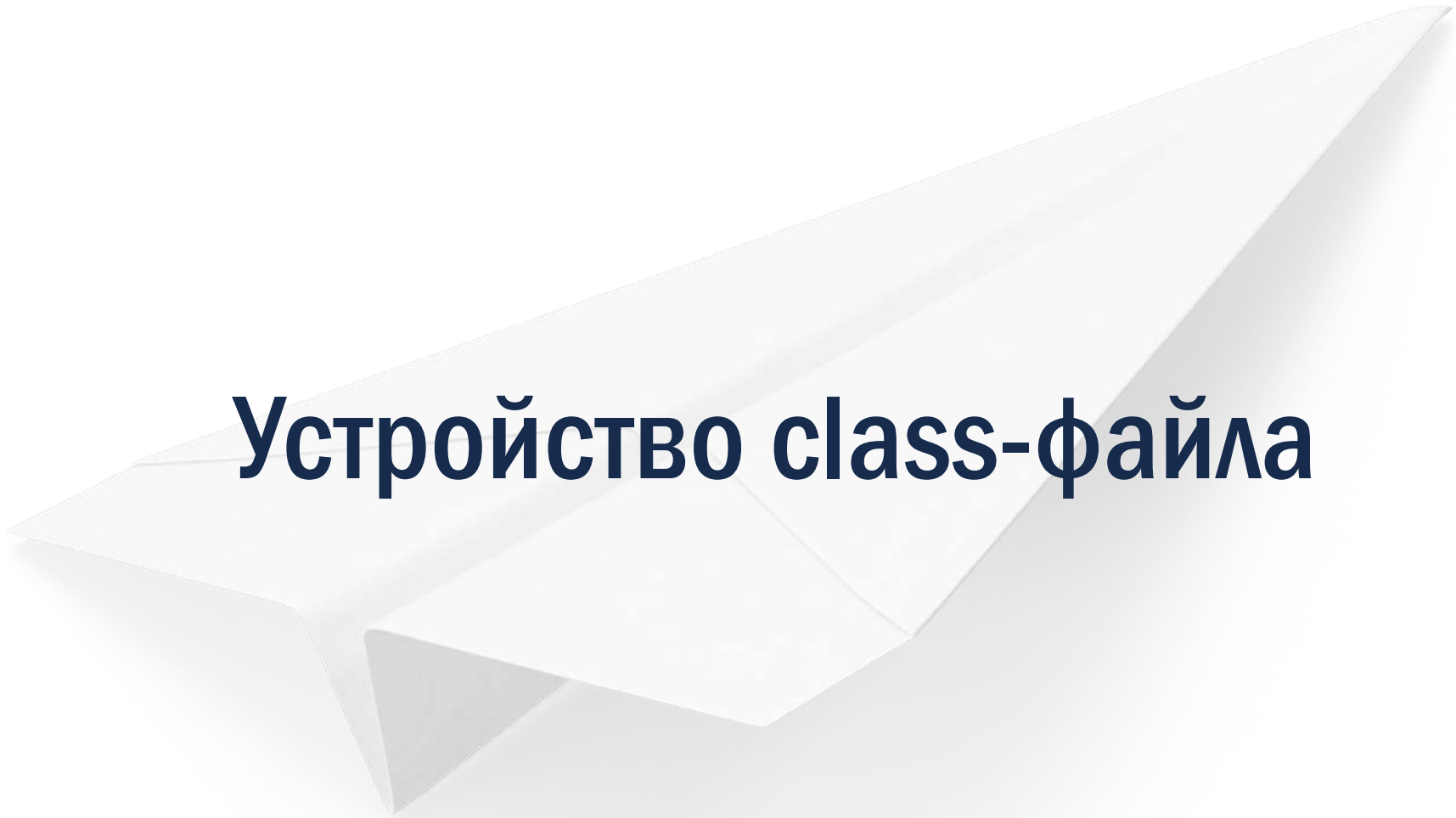



Runtime Data Area
Method Area
Heap Area
Stack Area
Thread #1
Thread #2
PC Register
Thread #1
Thread #2
Code Cache
Native Method Stack


Execution Engine
Interpreter
JIT Compiler
GC


Kotlin
Program.java
Компилятор
011000010
101010001
000101000
001010101
Program.class
JVM
(Java Virtual
Machine)
Windows
Linux
MacOS
Scala
JDK
Байткод


Program.java
Компилятор
011000010
101010001
000101000
001010101
Program.class
Class Loader
Loading
Linking
Initialization
Runtime Data Area
Method Area
Heap Area
Stack Area
Thread #1
Thread #2
PC Register
Thread #1
Thread #2
Code Cache
Native Method Stack
Execution Engine
Interpreter
JIT Compiler
GC
Native Method Interface (JNI)
Native Method Library



Устройство class-файла

Class файл

1. magic
2. minor_version, major_version
3. constant_pool_count и constant_pool
4. access_flags
5. this_class
6. super_class
7. interfaces_count и interfaces
8. fields_count и fields
9. methods_count и methods
10. attributes_count и attributes

4.1. The ClassFile Structure

A class file consists of a single ClassFile structure:

```
ClassFile {
    u4             magic;
    u2             minor_version;
    u2             major_version;
    u2             constant_pool_count;
    cp_info        constant_pool[constant_pool_count-1];
    u2             access_flags;
    u2             this_class;
    u2             super_class;
    u2             interfaces_count;
    u2             interfaces[interfaces_count];
    u2             fields_count;
    field_info     fields[fields_count];
    u2             methods_count;
    method_info    methods[methods_count];
    u2             attributes_count;
    attribute_info attributes[attributes_count];
}
```

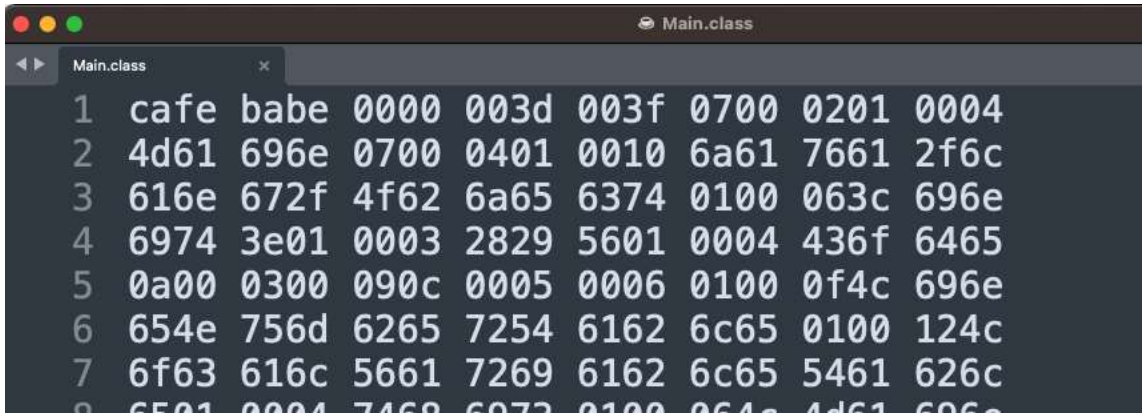
https://docs.oracle.com/javase/specs/jvms/se21/html/jvms-4.html

Header + Version

- Magic Number: 4 байта, уникальное значение (hex: 0xCAFEBABE), идентифицирующее файл как файл класса Java.
- Версия формата: 2 байта, указывает на версию формата class-файла.

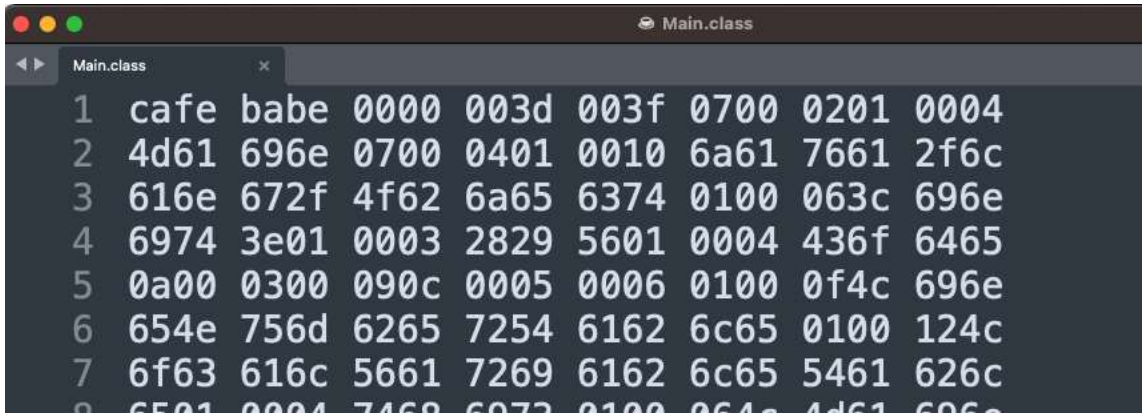
Header + Version

- Magic Number: 4 байта, уникальное значение (hex: 0xCAFEBABE), идентифицирующее файл как файл класса Java.
- Версия формата: 2 байта, указывает на версию формата class-файла.



Header + Version

- Magic Number: 4 байта, уникальное значение (hex: 0xCAFEBABE), идентифицирующее файл как файл класса Java.
- Версия формата: 2 байта, указывает на версию формата class-файла.



Error: LinkageError occurred while loading main class com.example.MainClass
java.lang.UnsupportedClassVersionError: com/example/MainClass has been compiled by a more recent version of the Java Runtime (class file version 55.0), this version of the Java Runtime only recognizes class file versions up to 52.0

Constant pool

Константа – тип, значение и номер (ссылка).

Constant pool

Константа – тип, значение и номер (ссылка).

Базовые типы:

- Integer
- Long
- Float
- Double
- UTF8

Constant pool

Константа – тип, значение и номер (ссылка).

Ссылочные типы:

- String
- NameAndType
- Class
- Fieldref
- MethodRef, InterfaceMethodRef

Access flags, this, super

Table 4.1-B. Class access and property modifiers

Flag Name	Value	Interpretation
ACC_PUBLIC	0x0001	Declared `public`; may be accessed from outside its package.
ACC_FINAL	0x0010	Declared `final`; no subclasses allowed.
ACC_SUPER	0x0020	Treat superclass methods specially when invoked by the *invokespecial* instruction.
ACC_INTERFACE	0x0200	Is an interface, not a class.
ACC_ABSTRACT	0x0400	Declared `abstract`; must not be instantiated.
ACC_SYNTHETIC	0x1000	Declared synthetic; not present in the source code.
ACC_ANNOTATION	0x2000	Declared as an annotation interface.
ACC_ENUM	0x4000	Declared as an `enum` class.
ACC_MODULE	0x8000	Is a module, not a class or interface.

```
this_class: #13                         // course/lecture2/example/ConstantPoolExample
super_class: #2                         // java/lang/Object
```

Fields

FieldType term	Type	Interpretation
B	`byte`	signed byte
C	`char`	Unicode character code point in the Basic Multilingual Plane, encoded with UTF-16
D	`double`	double-precision floating-point value
F	`float`	single-precision floating-point value
I	`int`	integer
J	`long`	long integer
L *ClassName* ;	`reference`	an instance of class *ClassName*
S	`short`	signed short
Z	`boolean`	`true` or `false`
[	`reference`	one array dimension

Methods

Дескрипторы.

Шаблон - ([param1[param2[...]]])returnValue

- (BB)I
- (I)V
- java/lang/Object."<init>":()V

Attributes

- ConstantValue
- Code
- StackMapTable
- BootstrapMethods
- NestHost
- NestMembers
- PermittedSubclasses

```
attribute_info {
    u2 attribute_name_index;
    u4 attribute_length;
    u1 info[attribute_length];
}
```

Code!

```
Code_attribute {
    u2 attribute_name_index;
    u4 attribute_length;
    u2 max_stack;
    u2 max_locals;
    u4 code_length;
    u1 code[code_length];
    u2 exception_table_length;
    {   u2 start_pc;
        u2 end_pc;
        u2 handler_pc;
        u2 catch_type;
    } exception_table[exception_table_length];
    u2 attributes_count;
    attribute_info attributes[attributes_count];
}
```

Class файл

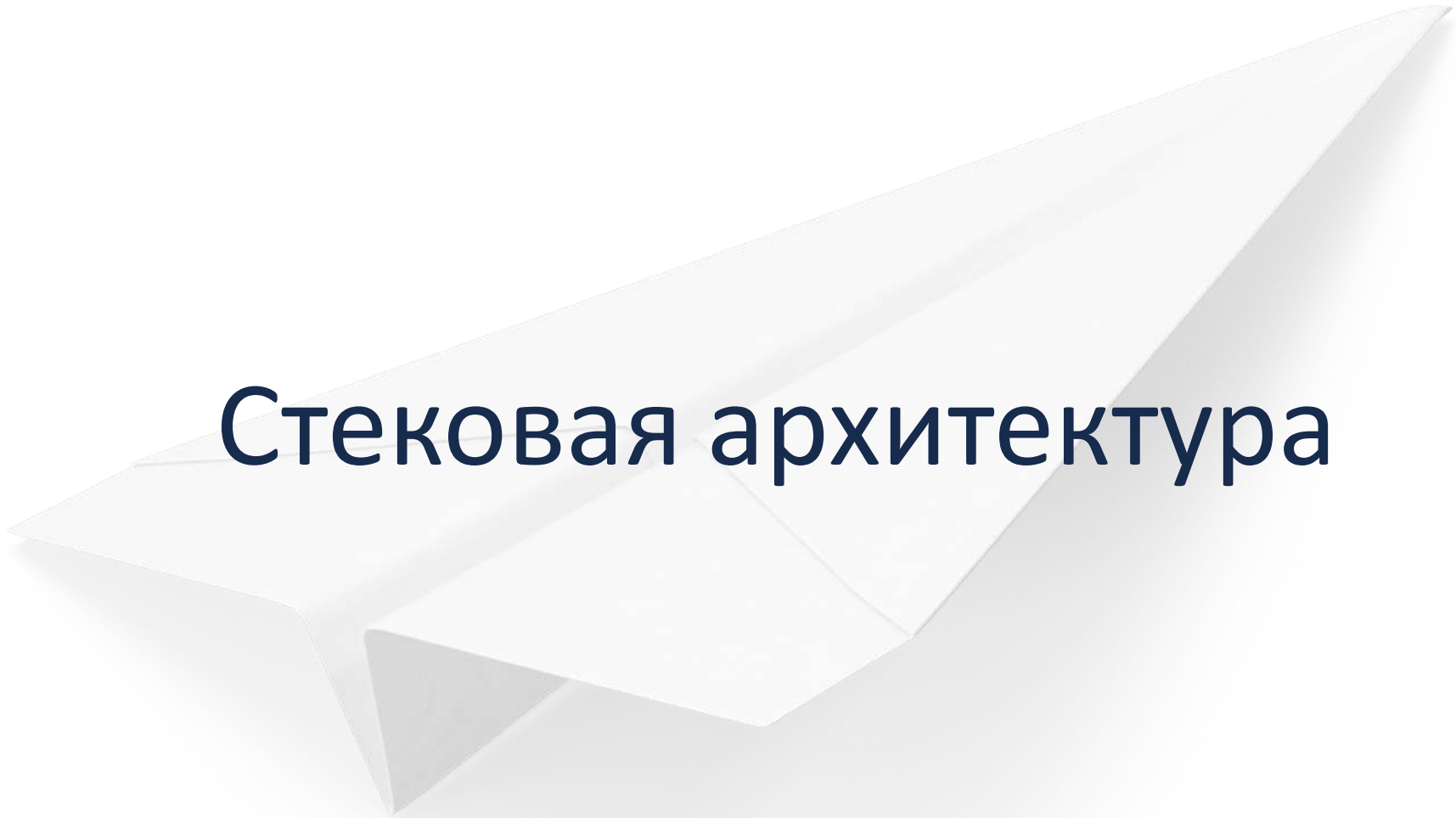
1. magic
2. minor_version, major_version
3. constant_pool_count и constant_pool
4. access_flags
5. this_class
6. super_class
7. interfaces_count и interfaces
8. fields_count и fields
9. methods_count и methods
10. attributes_count и attributes

4.1. The ClassFile Structure

A class file consists of a single ClassFile structure:

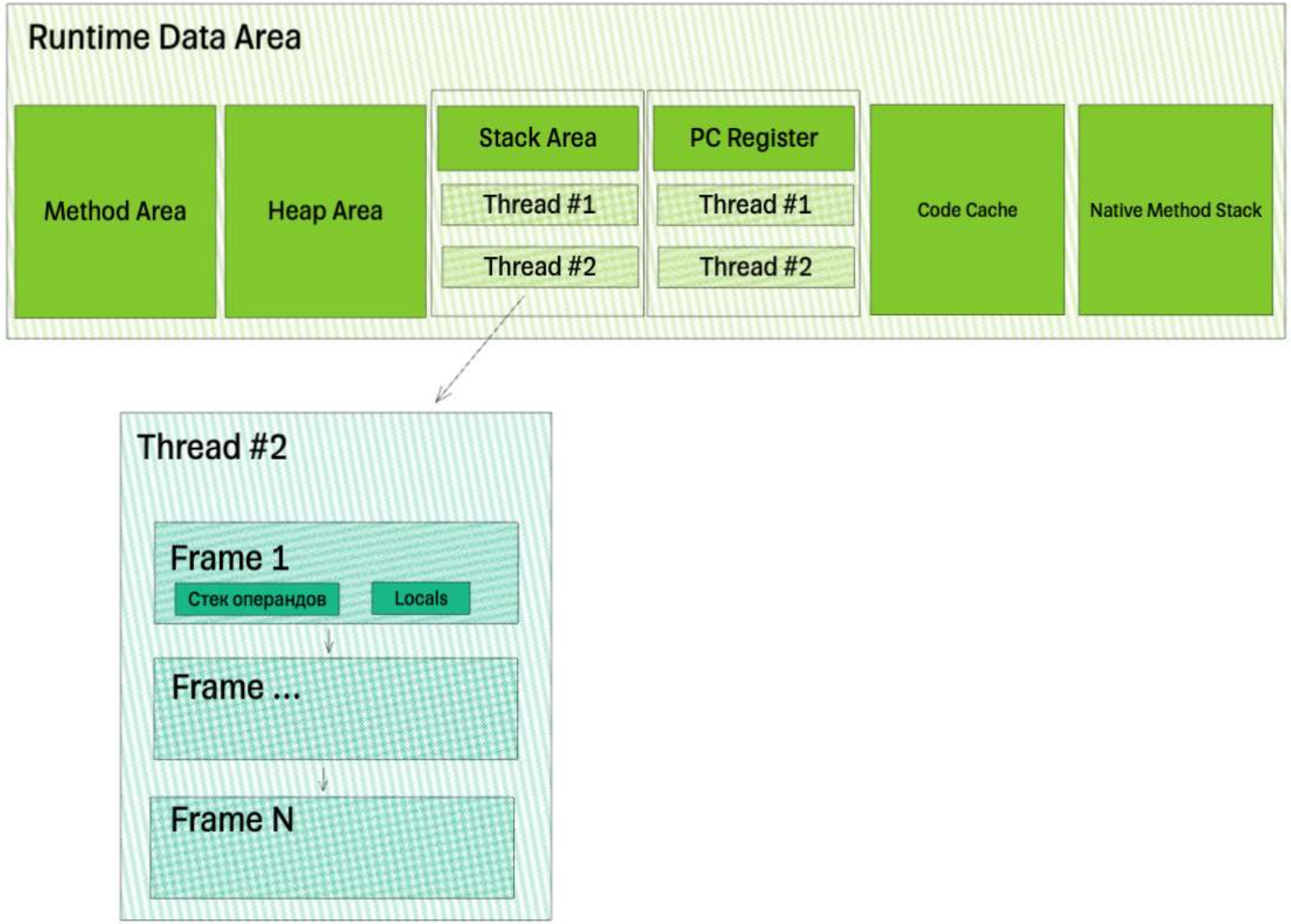
```
ClassFile {
    u4             magic;
    u2             minor_version;
    u2             major_version;
    u2             constant_pool_count;
    cp_info        constant_pool[constant_pool_count-1];
    u2             access_flags;
    u2             this_class;
    u2             super_class;
    u2             interfaces_count;
    u2             interfaces[interfaces_count];
    u2             fields_count;
    field_info     fields[fields_count];
    u2             methods_count;
    method_info    methods[methods_count];
    u2             attributes_count;
    attribute_info attributes[attributes_count];
}
```

https://docs.oracle.com/javase/specs/jvms/se21/html/jvms-4.html



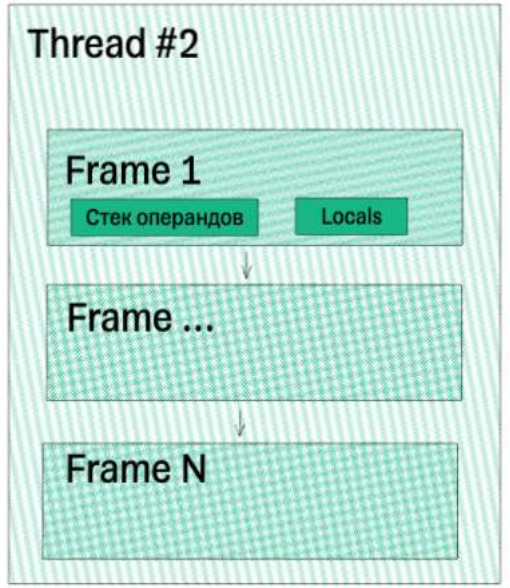
Стековая архитектура

Стековая архитектура



Стековая архитектура

1. **Стек операндов:**
    1. **Цель:** Хранение промежуточных результатов вычислений и операндов для выполнения инструкций.
    2. **Использование:** Операнды для большинства инструкций загружаются на стек операндов перед выполнением операции, и результаты сохраняются обратно на стек. Это обеспечивает временное хранение промежуточных данных.
2. **Локальные переменные (locals):**
    1. **Цель:** Хранение локальных переменных метода и аргументов метода.
    2. **Использование:** Каждая локальная переменная в методе имеет свое место в массиве locals. Эти переменные используются для временного хранения значений внутри метода, и к ним можно обращаться напрямую по их индексам.

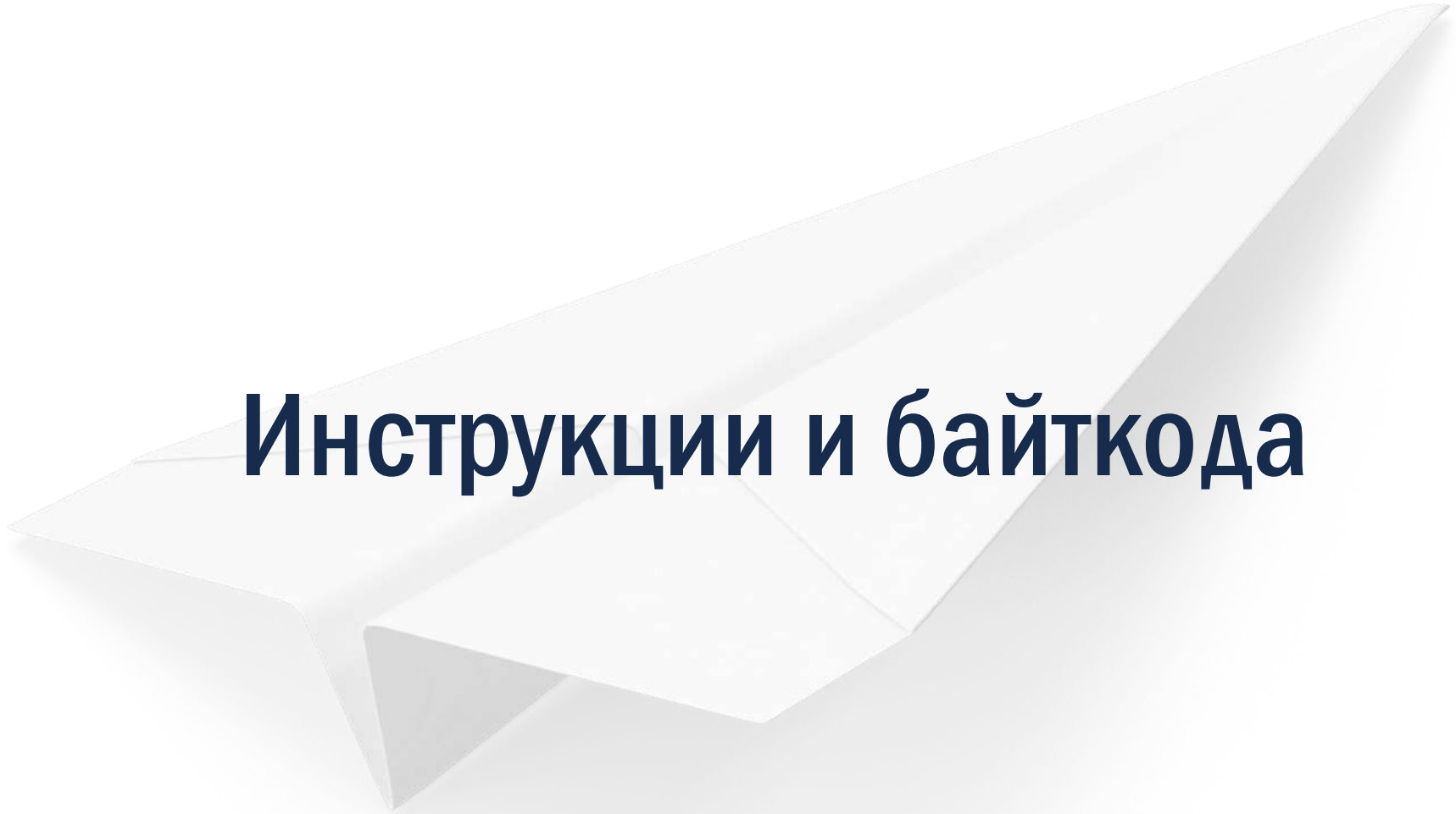


Стековая архитектура

```java
public class EasyAddMain {
    public static void main(String[] args) {
        int arg1 = 2;
        int arg2 = 7;
        int sum = addNumbers(arg1, arg2);

        System.out.println(sum);
    }

    1 usage
    private static int addNumbers(int arg1, int arg2) {
        int sum;
        sum = arg1 + arg2;
        return sum;
    }
}
```

```
private static int addNumbers(int, int);
  descriptor: (II)I
  flags: (0x000a) ACC_PRIVATE, ACC_STATIC
  Code:
    stack=2, locals=3, args_size=2
       0: iload_0
       1: iload_1
       2: iadd
       3: istore_2
       4: iload_2
       5: ireturn
```

0: iload_0 - загрузка значения переменной arg1 на стек операндов
1: iload_1 - загрузка значения переменной arg2 на стек операндов
2: iadd - сложение; результат остается на стеке операндов
3: istore_2 - сохранение результата в локальную переменную sum
4: iload_2 - загрузка значения переменной sum на стек операндов
5: ireturn - возврат результата



Инструкции и байткода

Стековая архитектура

1.Загрузка и сохранение значений:

1. iload, fload, aload: Загрузка значения с локальной переменной на стек операндов (int, float, ссылка).
2. istore, fstore, astore: Сохранение значения со стека операндов в локальную переменную (int, float, ссылка).
3. iconst, fconst: Загрузка значения константы на стек операндов (int,float)

Стековая архитектура

1.Загрузка и сохранение значений:

1. iload, fload, aload: Загрузка значения с локальной переменной на стек операндов (int, float, ссылка).
2. istore, fstore, astore: Сохранение значения со стека операндов в локальную переменную (int, float, ссылка).
3. iconst, fconst: Загрузка значения константы на стек операндов (int,float)

2.Арифметические операции:

1. iadd, isub, imul, idiv: Сложение, вычитание, умножение, деление для целых чисел.
2. fadd, fsub, fmul, fdiv: Сложение, вычитание, умножение, деление для чисел с плавающей точкой.

Стековая архитектура

1.Загрузка и сохранение значений:

1. iload, fload, aload: Загрузка значения с локальной переменной на стек операндов (int, float, ссылка).
2. istore, fstore, astore: Сохранение значения со стека операндов в локальную переменную (int, float, ссылка).
3. iconst, fconst: Загрузка значения константы на стек операндов (int,float)

2.Арифметические операции:

1. iadd, isub, imul, idiv: Сложение, вычитание, умножение, деление для целых чисел.
2. fadd, fsub, fmul, fdiv: Сложение, вычитание, умножение, деление для чисел с плавающей точкой.

3.Управление потоком выполнения:

1. if<condition>, goto: Условные и безусловные переходы.
2. return: Возврат из метода.
3. athrow: Выброс исключения.
4. jsr, ret: Подпрограммы и возврат из подпрограммы.

Стековая архитектура

1.Загрузка и сохранение значений:

1. iload, fload, aload: Загрузка значения с локальной переменной на стек операндов (int, float, ссылка).
2. istore, fstore, astore: Сохранение значения со стека операндов в локальную переменную (int, float, ссылка).
3. iconst, fconst: Загрузка значения константы на стек операндов (int,float)

2.Арифметические операции:

1. iadd, isub, imul, idiv: Сложение, вычитание, умножение, деление для целых чисел.
2. fadd, fsub, fmul, fdiv: Сложение, вычитание, умножение, деление для чисел с плавающей точкой.

3.Управление потоком выполнения:

1. if<condition>, goto: Условные и безусловные переходы.
2. return: Возврат из метода.
3. athrow: Выброс исключения.
4. jsr, ret: Подпрограммы и возврат из подпрограммы.

4.Работа с массивами:

1. iaload, iastore: Загрузка и сохранение элемента массива целых чисел.
2. aaload, aastore: Загрузка и сохранение элемента массива ссылок.

Стековая архитектура

5. **Работа с объектами:**
    1. new: Создание нового экземпляра класса.
    2. getfield, putfield: Чтение и запись значений полей объекта.

Стековая архитектура

5. **Работа с объектами:**
    1. new: Создание нового экземпляра класса.
    2. getfield, putfield: Чтение и запись значений полей объекта.

6.Вызов методов:

1. invokevirtual, invokespecial, invokestatic: Вызов методов (виртуальных, специальных, статических).

Стековая архитектура

5. **Работа с объектами:**
    1. new: Создание нового экземпляра класса.
    2. getfield, putfield: Чтение и запись значений полей объекта.

6.Вызов методов:

1. invokevirtual, invokespecial, invokestatic: Вызов методов (виртуальных, специальных, статических).

7.Конвертация типов:

1. i2f, i2d, f2i, f2d, d2i, d2f: Преобразование между типами (int, float, double).

Стековая архитектура

5. **Работа с объектами:**
    1. new: Создание нового экземпляра класса.
    2. getfield, putfield: Чтение и запись значений полей объекта.

6.Вызов методов:

1. invokevirtual, invokespecial, invokestatic: Вызов методов (виртуальных, специальных, статических).

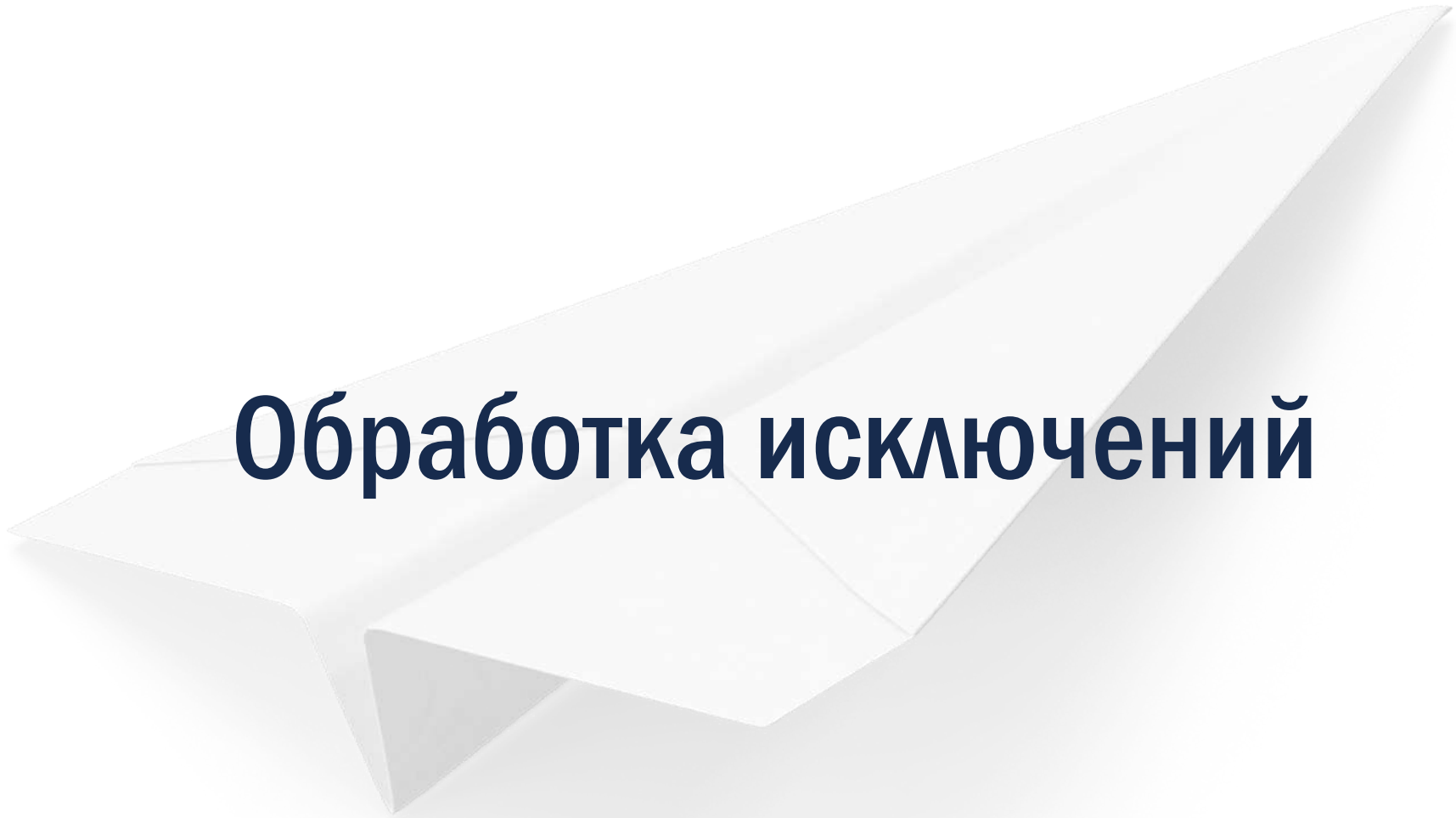
7.Конвертация типов:

1. i2f, i2d, f2i, f2d, d2i, d2f: Преобразование между типами (int, float, double).

8.Другие операции:

1. nop: Пустая операция (no operation).
2. pop, pop2: Удаление значения(й) со стека операндов.

https://en.wikipedia.org/wiki/List_of_Java_bytecode_instructions



Обработка исключений

```java
public class ExceptionExample {
    public static void main(String[] args) {
        System.out.println(someMethod());
    }

    1 usage
    public static int someMethod() {
        try {
            int arg1 = 5 / 0;
            return arg1;
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return 0;
    }
}
```

```
public static int someMethod();
  descriptor: ()I
  flags: (0x0009) ACC_PUBLIC, ACC_STATIC
  Code:
    stack=2, locals=1, args_size=0
       0: iconst_5
       1: iconst_0
       2: idiv
       3: istore_0
       4: iload_0
       5: ireturn
       6: astore_0
       7: getstatic     #7                  // Field java/lang/System.out:Ljava/io/PrintStream;
      10: aload_0
      11: invokevirtual #27                 // Method java/lang/Exception.getMessage:()Ljava/lang/String;
      14: invokevirtual #31                 // Method java/io/PrintStream.println:(Ljava/lang/String;)V
      17: iconst_0
      18: ireturn
    Exception table:
       from    to  target type
           0     5     6   Class java/lang/Exception
```

```
Compiled from "ExmapleMaing.java"
public class course.lecture2.example.ExmapleMaing {
  public course.lecture2.example.ExmapleMaing();
    Code:
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."<init>":()V
       4: return

  public static void main(java.lang.String[]);
    Code:
       0: getstatic     #7                  // Field java/lang/System.out:Ljava/io/PrintStream;
       3: ldc           #13                 // String Hello World!
       5: invokevirtual #15                 // Method java/io/PrintStream.println:(Ljava/lang/String;)V
       8: return

  public java.lang.Integer getValue(java.lang.String);
    Code:
       0: iconst_5
       1: invokestatic  #21                 // Method java/lang/Integer.valueOf:(I)Ljava/lang/Integer;
       4: areturn
}
```

Следующая лекция – Устройство JVM

- ClassLoader-ы
- GC
- Class Initialization (подробно)
- Профилирование

В некоторых сценариях, особенно при генерации или оптимизации кода, может потребоваться поддерживать определенное выравнивание инструкций. Вставка инструкции nop может быть одним из способов достижения этой цели, так как она не влияет на логику программы, но может влиять на распределение инструкций в памяти.

Другой сценарий использования nop связан с временными патчами или заглушками. Например, если вам нужно быстро внести изменения в байткод, но не хотите изменять логику программы, вы можете временно заменить часть кода на nop. После этого вы можете вернуться к изменениям, когда это станет удобным.

Оставь обратную связь

https://polls.tinkoff.ru/s/clnd1f38w001v01kfbin9ebvq